C# 프로그래밍

- 15 주차 (2강)

SEJONG UNIVERSITY

람다식 소개
문 형식의 람다식
전용대리자 : Func와 Action
식 본문 멤버

람다식 (Lambda Expression)

- 람다식은 람다 계산법이라는 수학 이론에 사용하는 식
- 수학 이론에 그치지 않고 프로그래밍 언어에 도입
- 람다식은 익명 메소드를 만들기 위해 사용
  - ✓ 람다식으로 만드는 익명 메소드는 무명함수(Anonymous Function)이라 부름
  - ✓ 람다식 선언은 입력 연산자 '=>'를 중심으로 왼편은 매개변수가, 오른편엔 식이 위치

사용 형식 :

```
( 매개변수_목록 ) => 식
```

사용 예제 :

```
delegate int Calculate(int a, int b);

static void Main(string[] args)
{
    Calculate calc = (int a, int b) => a + b;
}
```

익명 메소드를 만들려면 대리자가 반드시 필요

두 개의 int 형식 매개변수 a, b를 받아 이 둘의 합을 반환하는 익명메소드를 람다식으로 구현

형식 유추 (Type Inference)

- 대리자 선언 코드로부터 람다식이 만드는 익명 메소드의 매개변수의 형식을 유추
  - ✓ 람다식을 작성할 때 매개변수 형식을 제거하여 간편하게 나타낼 수 있음

사용 예제 : 형식 유추 기반의 람다식

```
delegate int Calculate(int a, int b);

static void Main(string[] args)
{
    Calculate calc = (a, b) => a + b;
    Console.WriteLine($"3 + 4 : {calc(3, 4)}"); // 출력 : 3 + 4 : 7
}
```

Calculator 대리자의 선언 코드로부터 해당 람다식의 매개변수의 형식을 유추

사용 예제 : delegate 키워드를 이용한 익명 메소드

```
delegate int Calculate(int a, int b);

static void Main(string[] args)
{
    Calculate calc = delegate (int a, int b)
                     {
                        return a + b;
                     };
}
```

람다식과 비교하여 코드가 길어짐

문 형식의 람다식

- '=>' 연산자의 오른편에 식 대신에 if 조건문과 같은 문장을 사용
  - ✓ '=>' 연산자의 오른편에 식 대신 {와 }로 둘러싸인 코드 블록이 위치
  - ✓ 반환 형식이 없는 무명 함수도 만들 수 있음

사용 형식 :

```
( 매개변수_목록 ) => {
                        문장1;
                        문장2;
                        문장3;
                        ...
                    }
```

사용 예제 :

```
delegate void DoSomething();
// ...
static void Main(string[] args)
{
    DoSomething DoIt = ( ) =>
            {
                Console.WriteLine("출력1");
                Console.WriteLine("출력2");
                Console.WriteLine("출력3");
            };
    DoIt();
}
```

매개변수가 없는 경우

문장 형식의 람다식은 {와 }로 둘러쌈

문 형식의 람다식 예제코드

```
delegate string Concatenate(string[] args);

static void Main(string[] args)
{
    Concatenate concat =
        (arr) =>
        {
            string result = "";
            foreach (string s in arr)
                result += s;

            return result;
        };

    Console.WriteLine(concat(args));
}
```

출력 결과:
>StatementLambda 출력을 확인 합니다.
출력을확인합니다.

무명함수를 위한 전용 대리자 : Func와 Action

- 무명 함수를 만들기 위해 매번 대리자를 선언해야 하는 번거로움이 있음
- .NET은 Func와 Action 대리자를 미리 정의하여 이 번거로움에 관한 문제 해결
- Func 대리자
  - ✓ 결과를 반환하는 익명 메소드 또는 무명 함수를 참조
  - ✓ 입력 매개변수가 없는 것부터 16개에 이르는 것까지 버전이 다양

Func 대리자의 형식 매개변수 중 가장 마지막에 있는 것이 반환 형식

```
public delegate TResult Func<out TResult>(); // 입력 매개변수는 없고, 반환형식만 존재
public delegate TResult Func<in T, out TResult>(T arg); // 입력 매개변수 한개
public delegate TResult Func<in T1, in T2, out TResult>(T1 arg1, T2 arg2); // 입력 매개변수 두개
…
public delegate TResult Func<in T1, in T2, … , in T15, in T16, out TResult>(T1 arg1, T2 arg2, … ,
T15 arg15, T16 arg16); // 입력 매개변수 열여섯개
```

매개변수 한정자	메소드 안에서 읽기	메소드 안에서 쓰기
in	가능	불가능
out	가능	반드시 해야함

Func 대리자 사용 예제

- 매개변수가 없는 버전, Func<TResult>의 사용 예

```
Func<int> func1 = () => 10;         // 입력 매개변수는 없으며, 무조건 10을 반환
Console.WriteLine($"{func1()}");    // 10 출력
```

- 매개변수가 하나 있는 버전, Func<T1, TResult>의 사용 예

```
Func<int, int> func2 = (x) => x * 2; // 입력 매개변수는 int 형식 하나, 반환 형식 int
Console.WriteLine($"{func2(4)}");    // 8 출력
```

- 매개변수가 두 개 있는 버전, Func<T1, T2, TResult>의 사용 예

```
Func<double, double, double> func3 = (x, y) => x / y; // 입력 매개변수는 double 형식 둘,
                                                      // 반환 형식 double
Console.WriteLine($"{func3(24, 7)}");                 // 3.4285714285714284 출력
```

Func 대리자 사용 예제코드

```
static void Main(string[] args)
{
    Func<int> func1 = () => 10;
    Console.WriteLine($"func1() : {func1()}");

    Func<int, int> func2 = (x) => x * 2;
    Console.WriteLine($"func2(4) : {func2(4)}");

    Func<double, double, double> func3 = (x, y) => x / y;
    Console.WriteLine($"func3(22, 7) : {func3(24, 7)}");
}
```

출력 결과:

```
func1() : 10
func2(4) : 8
func3(22, 7) : 3.4285714285714284
```

무명함수를 위한 전용 대리자 : Func와 Action

- Action 대리자
  - ✓ 결과를 반환하지 않는 익명 메소드 또는 무명 함수를 참조
  - ✓ 입력 매개변수가 없는 것부터 16개에 이르는 것까지 버전이 다양

Action 대리자의 형식 매개변수는 모두 입력 매개변수를 위해 선언

```
public delegate void Action<>(); // 입력 매개변수가 없음
public delegate void Action<in T>(T arg); // 입력 매개변수 한 개
public delegate void Action<in T1, in T2>(T1 arg1, T2 arg2); // 입력 매개변수 두 개
…
public delegate void Action<in T1, in T2, … , in T15, in T16>(T1 arg1, T2 arg2, … , T15 arg15, T16
arg16); // 입력 매개변수 열 여섯개
```

- 매개변수가 없는 버전, Action<>의 사용 예

```
Action act1 = () => Console.WriteLine("Action()");
act1();
```

Action 대리자 사용 예제

- 매개변수가 하나 뿐인 버전, Action<T>의 사용 예

```
int result = 0;
Action<int> act2 = (x) => result = x * x;

act2(3);
Console.WriteLine($"result : {result}");
```

람다식 밖에서 선언한 result에 x*x의 결과를 저장

// 9 출력

- 매개변수가 두 개 있는 버전, Action<T1, T2>의 사용 예

```
Action<double, double> act3 = (x, y) =>
{
    double pi = x / y;
    Console.WriteLine($"Action<T1, T2>({x}, {y}) : {pi}");
};

act3(22.0, 7.0);
```

Action 대리자 사용 예제코드

```
static void Main(string[] args)
{
    Action act1 = () => Console.WriteLine("Action()");
    act1();

    int result = 0;
    Action<int> act2 = (x) => result = x * x;

    act2(3);
    Console.WriteLine($"result : {result}");

    Action<double, double> act3 = (x, y) =>
    {
        double pi = x / y;
        Console.WriteLine($"Action<T1, T2>({x}, {y}) : {pi}");
    };

    act3(22.0, 7.0);
}
```

출력 결과:

```
Action()
result : 9
Action<T1, T2>(22, 7) : 3.142857142857143
```

식으로 이루어지는 멤버

- 메서드, 속성, 인덱서, 생성자, 종료자 멤버의 본문을 식(Expression)로만 구현 가능
  - ✓ "식 본문 멤버(Expression-Bodied Member)"라고 부름

사용 형식:

```
멤버 => 식;
```

- 사용 예제를 통해 식 본문 멤버에 대해 구체적으로 살펴보면,
  - ✓ list 필드 하나를 가지는 FriendList 클래스 선언

```
class FriendList
{
    private List<string> list = new List<string>();

    // 여기에 나머지 멤버 구현

}
```

식 본문 멤버 사용 예제

- 사용 예제를 통해 식 본문 멤버에 대해 구체적으로 살펴보면,
  - ✓ Add()와 Remove() 메서드는 각각 list.Add()와 list.Remove() 메서드를 호출하는 식으로 구현

```
class FriendList
{
    // ...

    public void Add(string name) => list.Add(name);
    public void Remove(string name) => list.Remove(name);
}
```

  - ✓ 생성자와 종료자도 식으로 구현 가능

```
class FriendList
{
    // ...

    public FriendList() => Console.WriteLine("FriendList()");    // 생성자
    ~FriendList() => Console.WriteLine("~FriendList()");         // 종료자
}
```

식 본문 멤버 사용 예제

- 사용 예제를 통해 식 본문 멤버에 대해 구체적으로 살펴보면,
  - ✓ 읽기 전용 속성과 인덱서를 식으로 구현하는 예제코드 (get 키워드조차 생략 가능)

```
class FriendList
{
    public int Capacity => list.Capacity; // 읽기 전용 속성
    public string this[int index] => list[index]; // 읽기 전용 인덱서
}
```

  - ✓ 읽기/쓰기 모두 가능한 속성 또는 인덱서를 구현할 땐 get/set 키워드 명시적으로 기술해야함

```
class FriendList
{
    public int Capacity // 속성
    {
        get => list.Capacity;
        set => list.Capacity = value;
    }

    public string this[int index] // 인덱서
    {
        get => list[index];
        set => list[index] = value;
    }
}
```

식 본문 멤버 사용 예제코드

```
class FriendList
{
    private List<string> list = new List<string>();

    public void Add(string name) => list.Add(name);
    public void Remove(string name) => list.Remove(name);
    public void PrintAll() {
        foreach (var s in list)
            Console.WriteLine(s);
    }

    public FriendList() => Console.WriteLine("FriendList()");
    ~FriendList() => Console.WriteLine("~FriendList()");

    //public int Capacity => list.Capacity; // 읽기 전용
    public int Capacity {
        get => list.Capacity;
        set => list.Capacity = value; }

    //public string this[int index] => list[index]; // 읽기 전용
    public string this[int index] {
        get => list[index];
        set => list[index] = value; }
}
```

```
static void Main(string[] args)
{
    FriendList obj = new FriendList();
    obj.Add("Jane");
    obj.Add("John");
    obj.Add("David");
    obj.Remove("Jane");
    obj.PrintAll();

    Console.WriteLine($"{obj.Capacity}");
    obj.Capacity = 10;
    Console.WriteLine($"{obj.Capacity}");

    Console.WriteLine($"{obj[0]}");
    obj[0] = "Vincent";
    obj.PrintAll();
}
```

출력 결과:

```
FriendList()
John
David
4
10
John
Vincent
David
```

Thank you!